CSC/ECE 506: Architecture of Parallel Computers
Sample Final Examination with Answers

This was a 180-minute open-web test. You were allowed use your laptop, the textbook and any course notes that you may have. We were not allowed to communicate with any other student, by electronic or other means, during the test.

You were to answer five of the six questions. Each question was worth 20 points. If you answered all six questions, your five highest scores counted.

Question 1. Cache misses can be classified into four types: cold, capacity, conflict, and coherence (see Lecture 14). Conflict misses can further be classified as either true sharing or false sharing.

This problem should be solved using the MESI protocol.
Assume the following:
Direct-mapped cache organization
P1 and P2 each have exactly 2 cache lines
cache block size = 4 words
B1 and B2 are two memory blocks that map to the same cache line. They contain the data items W, X, Y, Z; and P, Q, R, S, respectively as shown below. Each data item is one word.

B1:

W	*X*	*Y*	*Z*

B2:

P	*Q*	*R*	*S*

You are given the following trace of memory accesses from two processors P1 and P2. Assume that the accesses occur strictly sequentially in the textual order shown below.

Action	Hit/Miss Type
P1: Read Z	cold
P2: Read W	cold
P1: Read W	hit
P1: Write Z	hit; invalidates B1 in P2's cache.
P2: Read X	coherence, false sharing
P1: Read P	cold
P2: Write P	cold; invalidates B2 in P1's cache
P2: Write S	hit
P2: Read Y	conflict
P1: Read R	coherence, false sharing
P1: Write Q	hit
P2: Read X	hit
P1: Read W	conflict
P1: Write X	hit; invalidates B1 in P2's cache
P1: Write P	conflict (also true sharing)
P2: Write Z	coherence, true sharing; invalidates B1 in P1's cache
P1: Read X	conflict
P2: Write R	conflict (also true sharing)
P2: Write X	conflict; invalidates B2 in P1's cache
P1: Write Z	coherence, true sharing

Many people thought that there were capacity misses in the trace. This can't be, since there are only two blocks in use, and there are two lines in the cache. All of those misses are therefore conflict misses, caused by the fact that the two blocks map to the same line.

Question 2. *(4 points each)* For each code fragment below, put a check mark ("√∀)below *all* the consistency models under which they are legal. For each one that is not legal, write "x". For partial credit, you may give a reason.

(a)

P_1:	$W(x)1$		$R(x)1$	$R(x)2$
P_2:		$W(x)2$		
P_3:			$R(x)2$	

Sequential	Causal	Processor	PRAM
√	√	√	√

Answer: Sequentially consistent, causally consistent, PRAM consistent, and processor consistent. Not strictly consistent because the writes of 2 is not seen immediately by P_1. Sequentially consistent, because all writes are seen in order.

(b)

P_1:	$W(x)1$		$R(x)1$	
P_2:		$R(x)1$	$R(x)2$	
P_3:		$W(x)2$	$R(x)2$	$R(x)1$

Sequential	Causal	Processor	PRAM
x	√	x	√

Answer: Causally consistent, PRAM consistent. Not strictly consistent because the writes of 2 is not seen immediately by P_1. Not sequentially consistent, because the writes of 1 and 2 are seen in different orders by P_2 and P_3. However, these writes are not causally related, so it is still causally consistent. Not processor consistent because it is not memory coherent.

(c)

P_1:	$W(x)1$		$W(x)2$	$R(x)3$	
P_2:		$R(x)1$		$R(x)2$	
P_3:			$W(x)3$	$R(x)2$	$R(x)1$

Sequential	Causal	Processor	PRAM
x	√	x	x

Answer: None of the writes are causally related, so it is causally consistent "by default." Does not satisfy any of the other consistency models, since the two writes by P_1 are seen in different orders by P_2 and P_3.

(d)

P_1:	$W(x)1$			$R(x)2$
P_2:		$R(x)1$		
P_3:			$W(x)2$	$R(x)2$

Sequential	Causal	Processor	PRAM
√	√	√	√

Answer: Strictly consistent, sequentially consistent, causally consistent, PRAM consistent, processor consistent.

(e)

P_1:			$R(x)3$	$R(x)1$	$R(x)2$
P_2 :	$W(x)1$		$W(x)2$		
P_3:	$R(x)1$	$W(x)3$	$R(x)3$	$R(x)2$	

Sequential	Causal	Processor	PRAM
x	x	x	√

Answer: PRAM consistent (only). Both writes by P_2 are seen in the same order by the other processors. However, memory coherence does not hold, because the three writes to x are seen in a different order by P_1 are P_3. Causal consistency does not hold, because the writes of 1 and 3 are causally related, and are seen by P_1 are P_3 in a different order. This also means that sequential consistency and strict consistency do not hold.

Question 3. Consider how a memory-based cache directory might be organized. *Note:* In doing the calculations below, be careful not to confuse bits with bytes! (1 byte = 2^3 bits)

Suppose a multiprocessor has 256 nodes, each of which has 128 MB (= 2^{27} bytes) of memory. Suppose that a cache line contains 32 bytes.

(a) *(2 points)* How many cache blocks are there per node?

Answer: $2^{27}/2^5 = 2^{22}$, or 4 "megablocks."

(b) *(3 points)* If we used the full bit-vector approach for organizing the cache directory, what *fraction* of main memory would have to be devoted to cache directories?

Answer: For each block, we would need to use 256 bits to record which of the 256 processors contained it. This means 2^{22} blocks $\times$ 2^8 bits = 2^{30} bits = 2^{27} bytes. Woops—100% of main memory would be consumed by cache directories!

(c) *(3 points)* Assume that the average block is cached in 2 nodes. If we used pointers instead of the full bit-vector approach, what fraction of memory would be devoted to the pointers? (Answer the fraction of memory that would be devoted to *pointers* alone, not including other data structures that would be needed to keep track of the sharers.)

Answer: Each pointer is 8 bits (1 byte) long. Each of the 2^{22} blocks requires two pointers, on average. Therefore, there will be 2^{23} bytes devoted to pointers, out of 2^{27} bytes altogether. Thus, 1/16 of main memory will be devoted to pointers.

(d) *(3 points)* Is the assumption of part (c) realistic? Explain.

Answer: No, it is not realistic. If the "average" block were cached at 2 nodes, then there would have to be twice as much cache memory as main memory in the system! Therefore, the answer of part (c) is a gross overestimate of the amount of memory that would be devoted to pointers.

(e) *(2 points)* In addition to the pointers themselves, what other information would need to be kept in a pointer-based cache directory in order to allow the system to locate all nodes caching a particular block?

Answer: We would need some sort of indication of how many nodes were sharing each particular block. This could be in the form of a count of the number of sharers, or a link field that pointed to the next sharer.

(f) Another way that a memory-based directory could be optimized is by organizing it as a cache.

(i) *(3 points)* What kind of information would serve as the "tags" of such a cache? How large would a tag be?

Answer: The block number, a 22-bit number, would serve as the tag.

(ii) *(2 points)* What kind of information would be in the "lines" of such a cache?

Answer: A list of nodes that cached the block. This could be in the form of a pointer to the head of a linked list.

(iii) *(2 points) In one sentence or less*, suggest how this cache might be organized so that it could be searched easily in software.

Answer: As a hash table. There are other possibilities …

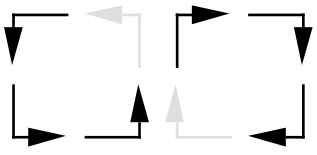
Question 4. Deadlock-free routing algorithms are the subject of this question.

(a) For each turn-model routing algorithm named below, draw a diagram of the turns allowed by it *(2 points each)*, tell whether it is deadlock free *(2 points each)*, and explain why or why not *(1 point each)*.

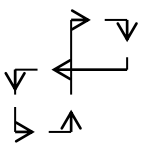
(i) East-first *Answer:* Deadlock free, because it is a rotation of west-first.

(ii) Positive-last *Answer:* This is the same as negative-first. Note that the last two turns made by positive-last are positive in both directions: from going right, it turns up; or from going up, it turns right. Therefore, it is deadlock free.

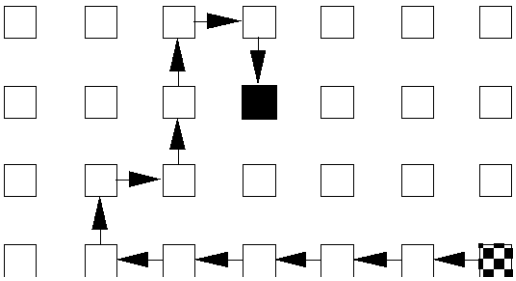
(b) I don't have a name for this algorithm, but tell if it is deadlock free *(2 points)* and why or why not *(1 point)*.



Answer: Not deadlock free; paths with cycles can easily be created, e.g.



(c) *(1 point each)* In which of the following routine algorithms would the path at the right be legal ? Answer "yes" or "no."



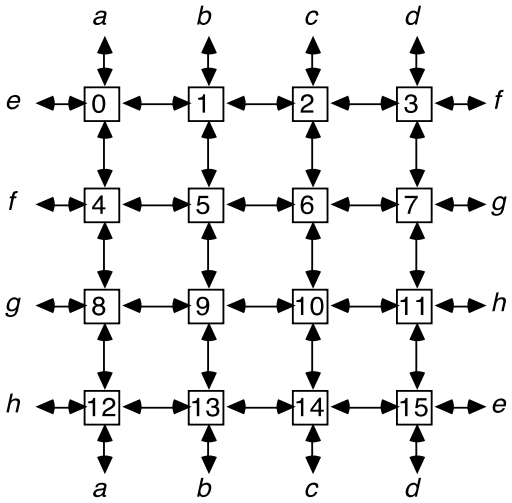
(i) Δx, Δy *Answer:* No

(ii) North-last *Answer:* No

(iii) Negative-first *Answer:* No

(iv) Positive-last *Answer:* No

(d) What is the shortest path(s) between node 0 and node 3 in this Illiac IV mesh (i.e., what nodes are traversed on the way from node 0 to node 3)? If a deadlock-free routing algorithm with two virtual channels is used, which channel will the message traverse at each hop on the way from node 0 to node 3?



Answer: The shortest path is either 0 → 4 → 3 or 0 → 15 → 3. In either case, the message goes over the high channel on the first hop and over the low channel on the second.

Question 5. *(1 point per line in (a)–(c); 2 pts. for part (d))* In a directory-based coherence protocol, one possible race condition is an *early-invalidation race* (Solihin §10.4.2, Lecture 23). This question involves tracing through the actions that occur when messages arrive in order, or out of order; and when an outstanding transaction buffer is in use, or not in use. All these scenarios involve Node A making a read request to a block, followed by Node B making a ReadX request to the same block. Neither node A nor node B is the home node.

(a) First, assume that all messages arrive at their destination in the order that they are sent. List the messages sent. (The first few are done for you.) Don't forget the ack's!

Sending node	Message*	Receiving node
A	Read	H
H	ReplyD	A
A	Ack	H
B	ReadX	H
H	Inv	A
A	InvAck	H
H	ReplyD	B
B	Ack	H

*Possible messages (Solihin p. 339) include

Read
ReadX
ReplyD
Ack
Nack
Inv
InvAck

(b) Same assumptions as (a), but an OTB is in use (the requester-assisted approach).

Sending node	Message	Receiving node	Anything buffered/removed from buffer? At which node(s)?

A	Read	H	Read buffered at A
H	ReplyD	A	Read removed from A's buffer
A	Ack	H	
B	ReadX	H	ReadX buffered at B
H	Inv	A	
A	InvAck	H	
H	ReplyD	B	ReadX removed from B's buffer
B	Ack	H	

(c) Same as (b), except that the invalidation message from H arrives at node A before the ReplyD message. (An OTB is in use.) List the messages in the order they arrive, *not the order in which they were sent*.

Sending node	Message	Receiving node	Anything buffered/removed from buffer? Where?
A	Read	H	Read buffered at A
B	ReadX	H	ReadX buffered at B
H	Inv	A	
A	Nack	H	(because it has a Read buffered)
H	ReplyD	A	Read removed from A's buffer
A	Ack	H	
H	Inv	A	(retries …)
A	InvAck	H	
H	ReplyD	B	ReadX removed from B's buffer

(d) If the home-centric scheme is use, why can't messages arrive out of order at node A or B?

Answer: With the home-centric scheme, the home will hold a request for a node until it knows that the node has acked all previous requests. Thus, messages can't arrive out of order.

Question 6. This question concerns calculation on an Illiac-IV type array processor with 16 processing elements (PE's). Each PE[i] has two scratchpad registers, $A[i]$ and $B[i]$, and a *routing register* $R[i]$. When a routing function is performed, the contents of $R[i]$ are transferred from the source PE to the destination PE. For example, when the +1 routing function ("**route** +1") is performed, the contents of $R[1]$ are transferred to $R[2]$, and the contents of $R[15]$ are transferred to $R[0]$, etc.

Sketch a program (in pseudo-C or a similar facsimile of a higher-level language) to calculate $B[i] = A[0] + A[1] + \cdots + A[15]$ for $i = 0, 1, 2, \ldots , 15$. (In other words, set each $B[i]$ to the sum of all of the $A[i]$.) Do *not* use more than eight invocations of routing functions.

Answer 6. Here are two possible programs.

$B[i] = A[i]$, $(0 \le i \le 15)$;
$R[i] = B[i]$, $(0 \le i \le 15)$;
route +1, $(0 \le i \le 15)$;
$B[i] = B[i] + R[i]$, $(0 \le i \le 15)$;
$R[i] = B[i]$, $(0 \le i \le 15)$;
route +1, $(0 \le i \le 15)$;
route +1, $(0 \le i \le 15)$;

```
B[i]  =  B[i] + R[i] ,      (0 ≤ i ≤ 15);
R[i]  =  B[i],              (0 ≤ i ≤ 15);
route  +4,                  (0 ≤ i ≤ 15);
B[i]  =  B[i] + R[i] ,      (0 ≤ i ≤ 15);
R[i]  =  B[i],              (0 ≤ i ≤ 15);
route  +4,                  (0 ≤ i ≤ 15);
route  +4,                  (0 ≤ i ≤ 15);
B[i]  =  B[i] + R[i] ,      (0 ≤ i ≤ 15);
```

or—

```
B[i]  =  A[i],                      (0 ≤ i ≤ 15);
R[i]  =  B[i],                      (0 ≤ i ≤ 15);
for j  = 1 to 3 do
        route  +1,                  (0 ≤ i ≤ 15);
        B[i]  =  B[i] + R[i] ,      (0 ≤ i ≤ 15);
end;

R[i]  =  B[i],                      (0 ≤ i ≤ 15);
for j  = 1 to 3 do
        route  +4,                  (0 ≤ i ≤ 15);
        B[i]  =  B[i] + R[i] ,      (0 ≤ i ≤ 15);
end;
```